**Names**: Adam Herd - 17364786, Jaime Kirk - 17922982
**Project Name**: Sentiment Analysis Online Chatbot
**Document**: Technical Guide
**Date**: 07/05/2021

## Table of Contents

**1. Abstract** 2

**2. Introduction** 2

**3. Motivation** 3

**4. Research** 4

**5. Design** 4

**6. Implementation** 6

**7. Problems Solved** 12

**8. Results** 13

**9. Future work** 15

**10. Appendix** 16

## 1. Abstract

Our project is a Sentiment Analysis online Chatbot. This chatbot is used to view products on Amazon (price, positive and negative reviews, link to product etc.) Users will interact with the chatbot by entering commands and then the chatbot will return a response if the command entered is valid or else a message stating that the command was not understood and to enter a valid command. All product information is gathered using Rainforest API which when called returns a list of all products that relate to the user's search query. We can then access this information and show it to the user.

It can also perform sentiment analysis on the reviews of whatever product you would like. This sentiment analysis is returned in the form of a pie chart that shows percentages regarding how positive and how negative the reviews were.

We are building deep learning models to be able to make predictions and classifications based on the data it has been trained on. For example, in the chatbot model, we are training it against a set of possible user inputs. Therefore, it will be able to predict when a user has entered such an input and return the appropriate response based on that.

For the sentiment analysis model, we are doing something similar. We are training a model against many reviews, each of which has a sentiment associated with it (positive or negative). The model will take in all the words from these reviews and be able to associate a sentiment with those words. A positive word will be a 1, and a negative word will be a 0. Therefore, when it comes to the reviews on Amazon, the model can look through the words and predict what class they belong to (positive or negative), and will return a decimal number which will represent how positive or how negative the review was; the higher the number, the more positive it is.

## 2. Introduction

The system we developed is a web-based Sentiment Analysis chatbot that allows users to easily access information about products on Amazon. The chatbot can be used by all age groups as it retrieves information about all products listed on Amazon so anyone interested in buying from Amazon can use our chatbot to retrieve product information before making their purchase. The aim was to make a bot that would gather all the information from Amazon's product pages and to return only what the user wants to save them time searching for information that can be returned instantly. Users can still access the full product pages by using chatbot to return the link of the product if they need further information or if they plan on making a purchase.

The chatbot allows users to lookup a list of products from their search query then select one of these products to get further information on such as price, product rating, get positive/negative reviews, get link to product and can perform sentiment analysis on the product reviews that return a pie chart with the results of the sentiment analysis. All this can be done easily by entering valid commands and most commands return instant responses. An example of this is the price command where if a user enters any valid version of the command such as “what is the price” then they will instantly receive the price for the product they have currently selected.

The user can also easily switch between searches by typing “end” which will reset their search and bring them back to the starting stage of the chatbot. This saves the user time in having to reload the webpage to start a new session or can be helpful if the user wants to compare information about certain products. The chatbot is displayed on a web page via a Python flask server.

# 3. Motivation

Our motivation for this project was based around a guest lecture we had in our second or third year. The lecture revolved around a system that was used during the 2014 World Cup to calculate the sentiment of an enormous number of tweets that were being sent out during the competition. This was our first glimpse into Sentiment Analysis. The project seemed really cool to us as we had not come across anything like this in our studies and have not since then either. So, we thought we would try our hand at it and see exactly what it entailed (from a beginner’s perspective).

We combined this with the idea of using a chatbot that you see very commonly on websites and apps nowadays. Learning how to build a bot that would respond with certain sentences in a natural way seemed like a good thing to learn and could come in useful down the line. It also allowed us to explore Deep Neural Networks.

Building a chatbot was interesting to us as we see them a lot using the internet for various purposes such as tech support on certain websites, placing orders for products etc. They are very flexible in terms of what you can build and use one for. So we felt that in learning to build our own would be a useful skill to us in the future. We wanted to build a machine learning chatbot as well as this is also a useful skill to know and is used widely nowadays. Like sentiment analysis, it was not something that we ever touched on in our course materials, so we were looking at this from a beginner’s point of view.

We saw an opportunity to combine the two of them in the form of this project. Also, it is quite different from our 3rd year project which we believe to be a good thing. Last year, we built an android app based around diet and fitness where users had a pedometer to track their steps and food options to log the food they had eaten in a day. This was a nice project and we learned a good deal from it, but this chatbot was something new to try and had the potential to teach us a great deal.

## 4. Research

There was quite a bit of research involved in this project due to the fact that we were working with various areas that we had never touched before (sentiment analysis, deep neural networks, natural language processing). Google was our main source for information. We needed to learn about all the areas mentioned as well as how to fit and train deep learning models using tflearn, how to parse the data these models require to train, how to link a JavaScript file with our python file and format our data to our website page, and more. We have linked to many resources that helped us along the way on our GitLab repo page and in the appendix of this document.

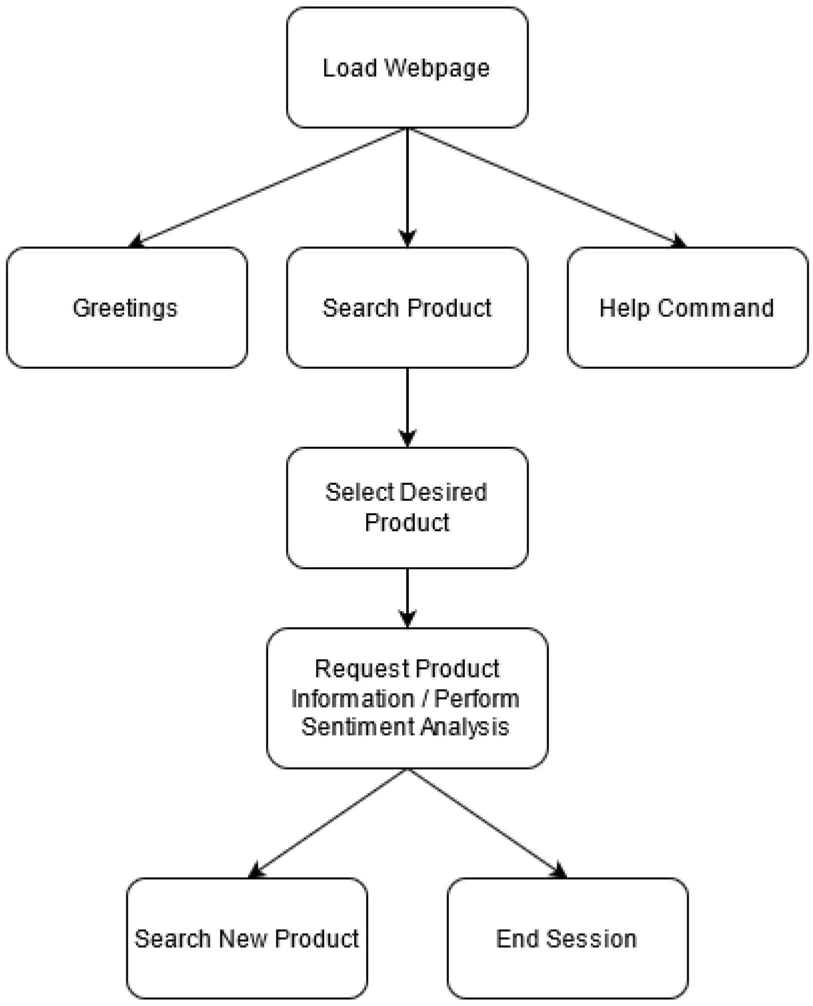
We also asked our supervisor for advice on the best approach and he organised a meeting with a research student that had worked on a project similar to ours. The research student gave us several ideas on what was the best way to do it like hosting the chatbot on a webpage instead of in some sort of application and also advised us to use Python over other programming languages so we could benefit from libraries such as Flask.

## 5. Design

Our project is used through a Python flask server, linking to a HTML page via a Javascript file. We have kept the webpage fairly simple as the user will be chatting with a bot so there is no need for fancy bells and whistles tagged on to that. The font we have chosen is Helvetica as this seems to be fairly popular because it is easy to read. We also added a little typewriter animation to the top of the page just for a bit of flair.

We have also tried to make the text contrast to the background a good bit so it stands out more. Each user message or bot message is also placed in a box which helps to differentiate them all and keeps things easy to follow as more text comes on the screen. We have had our test users say they like the colours and background as they make the text easy and clear to read. We also have a silly picture of a robot in the corner as we want to give the impression of a friendly and helpful bot.

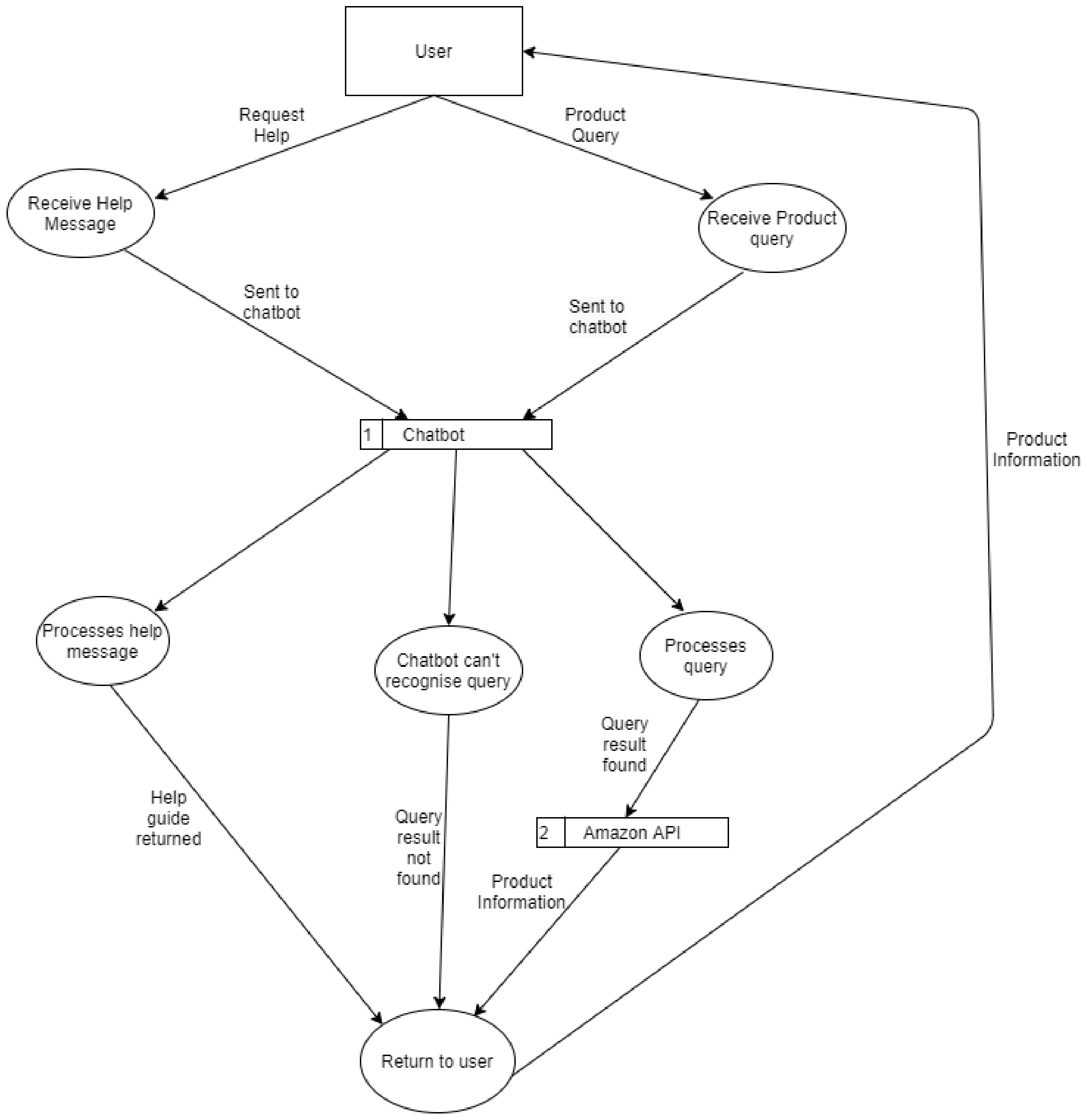
Below is a high-level design diagram of the system:



As we can see above, there are very few branching paths the user can take. Each command leads in to the next and then loops back on itself if the user wishes to search another product.

A typical run-through of the system would be as follows: The user will type their query into a text box at the bottom of the page and press enter. The Javascript function will display this message and also pass it on to the chatbot python file. Here, the message will be processed and passed to the model so the model can make a prediction for which class the message belongs to i.e the message “hi” will be classed as a greeting. Based on the classification the model gives us, certain actions will be taken. If the user is looking at a product and types “ratings” into the search box, then the classification will be made and in this case a call to the API will return the current ratings of the given product. The output from the model will come back to the Javascript file and be formatted so it can be outputted to the HTML page. And this continues for any and all queries the user makes.

The data-flow diagram below carries over from our function specification. Here we can see things in a bit more detail:



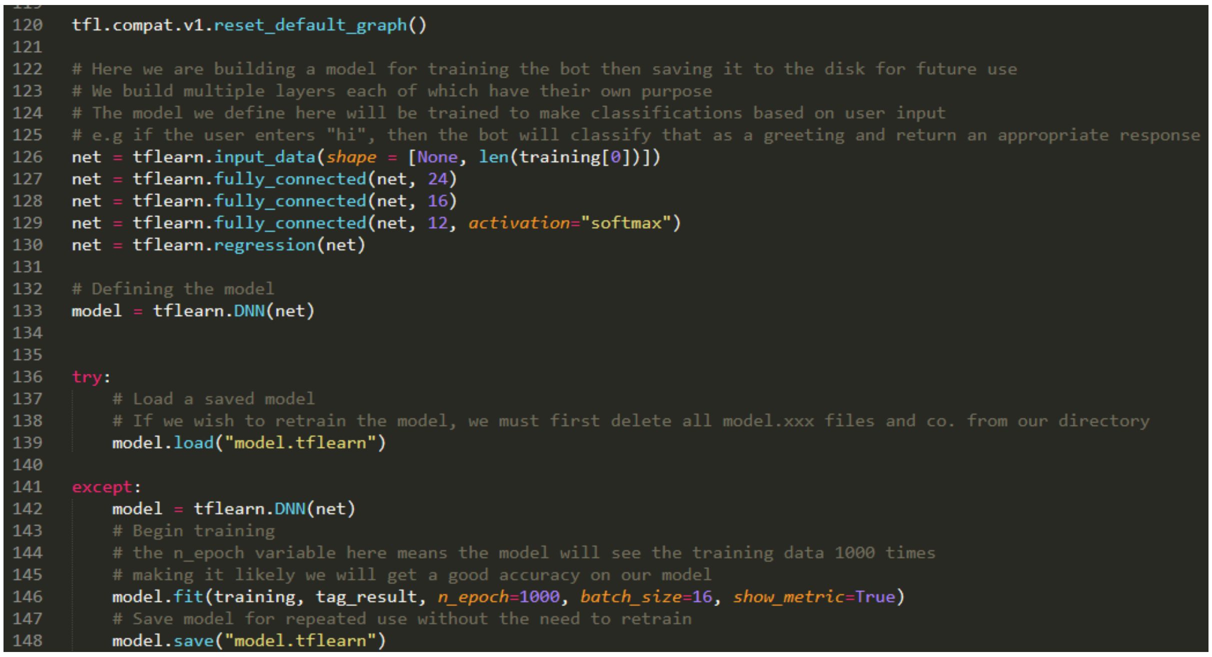
## 6. Implementation

To handle the communication between the user and the bot, we set up a JavaScript file that will interact with methods in our main Python file. The calculations for the bot’s response are handled in the Python file and once they are done, they are sent to the JavaScript file which then works to output the responses to the HTML web page.

The main Python file also handles the training of the bot. If there is no trained model present in the directory, then the file will take in the data we have in our JSON file and do the necessary workings to format the data so that tflearn can use it to train a model for the bot. This model will return a list of probabilities for which class it believes the user's message most belongs to, and we will find the largest probability and return a response from that class, and these responses are what is sent to the JavaScript file. However, sometimes we do not return a response from the JSON file and instead interact with the API based on the class the bot has deemed most appropriate.

In order to set up the flask server and do this training we first needed to format the data so TFlearn could use it, a tutorial on this was followed which can be found here: https://towardsdatascience.com/how-to-build-a-basic-chatbot-from-scratch-f63a2ccf5262 and also in the appendix. This tutorial was also used to set up the look of the site, with the boxes separating each block of text. Further research was done on this process which can be found through various links in the appendix.

Below is a sample of code. This piece of code handles the training of our bot to respond to user input. We will now talk through it.



```
120  tfl.compat.v1.reset_default_graph()
121
122  # Here we are building a model for training the bot then saving it to the disk for future use
123  # We build multiple layers each of which have their own purpose
124  # The model we define here will be trained to make classifications based on user input
125  # e.g if the user enters "hi", then the bot will classify that as a greeting and return an appropriate response
126  net = tflearn.input_data(shape = [None, len(training[0])])
127  net = tflearn.fully_connected(net, 24)
128  net = tflearn.fully_connected(net, 16)
129  net = tflearn.fully_connected(net, 12, activation="softmax")
130  net = tflearn.regression(net)
131
132  # Defining the model
133  model = tflearn.DNN(net)
134
135
136  try:
137      # Load a saved model
138      # If we wish to retrain the model, we must first delete all model.xxx files and co. from our directory
139      model.load("model.tflearn")
140
141  except:
142      model = tflearn.DNN(net)
143      # Begin training
144      # the n_epoch variable here means the model will see the training data 1000 times
145      # making it likely we will get a good accuracy on our model
146      model.fit(training, tag_result, n_epoch=1000, batch_size=16, show_metric=True)
147      # Save model for repeated use without the need to retrain
148      model.save("model.tflearn")
```

Line 120: Resets the graph stack so we can reuse the following lines from a clean state.

Line 126: The next handful of lines are all about building layers to our network. This first line feeds data to the network and creates a "shape". We are specifying how our input looks. The first argument is None, meaning the layer has no restrictions along that dimension. The second argument is one of our training lists. If we were to pass any data that did not fit this defined shape, then an exception would be raised.

Line 127-130: The next three layers are fully connected layers. The first argument they take is any previous layers, hence all layers assigned to a variable *net* and that same variable being passed as the first argument. The second argument decides on the number of units for these layers. This took some messing around to find what was best suited to our data and what returned the best results for us.

The final fully connected layer can be viewed as an output layer. Here, we have passed the number 12 as our units as there are twelve different types of classifications the bot can make (the JSON file has twelve different tags, which are the classes the bot will choose from). We have added “activation=softmax” here as well as this deals with classification. See the tflearn.org website for more on this. Link in appendix.

Line 133: Here we are defining our model using the DNN function and passing our layers via the *net* variable. This function sets up the deep learning model based on the layers we have previously defined.

Line 136-139: Here we are checking if there is already a model present in the directory. If so, then we will not proceed with training the model as there would be no need to do so and it would waste time. However, if there is no model present, then the exception would be triggered.

Line 141-148: If there is no model present in the directory, then this block of code will be executed. The *fit* function is used to begin our training. We pass our training data and label_results list as the first two arguments. The n_epoch argument specifies how many times the network will see the data. We have it set quite high at 1,000, but our training data is relatively small so the execution time is extremely small. Also, a higher epoch can increase our chances of the model having good accuracy (i.e its ability to accurately classify the user’s input).

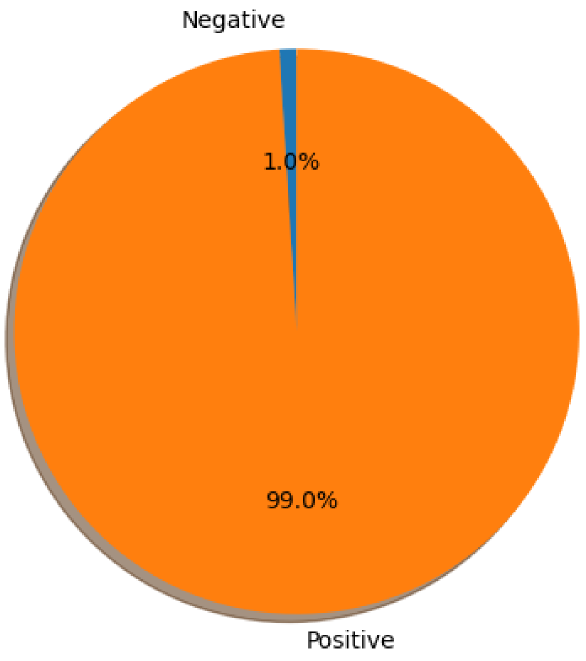
Lastly, we save the model to our directory under the name “model.tflearn”. It has the same name as the model we try to load in our try statement. This is so we can reuse it again and again without the need to train it again.

For the sentimentAnalysis.py file, we do much the same as the main bot file. We gather a bunch of data and format it so tflearn can train a model to be able to predict the sentiment of a review or bunch of reviews.

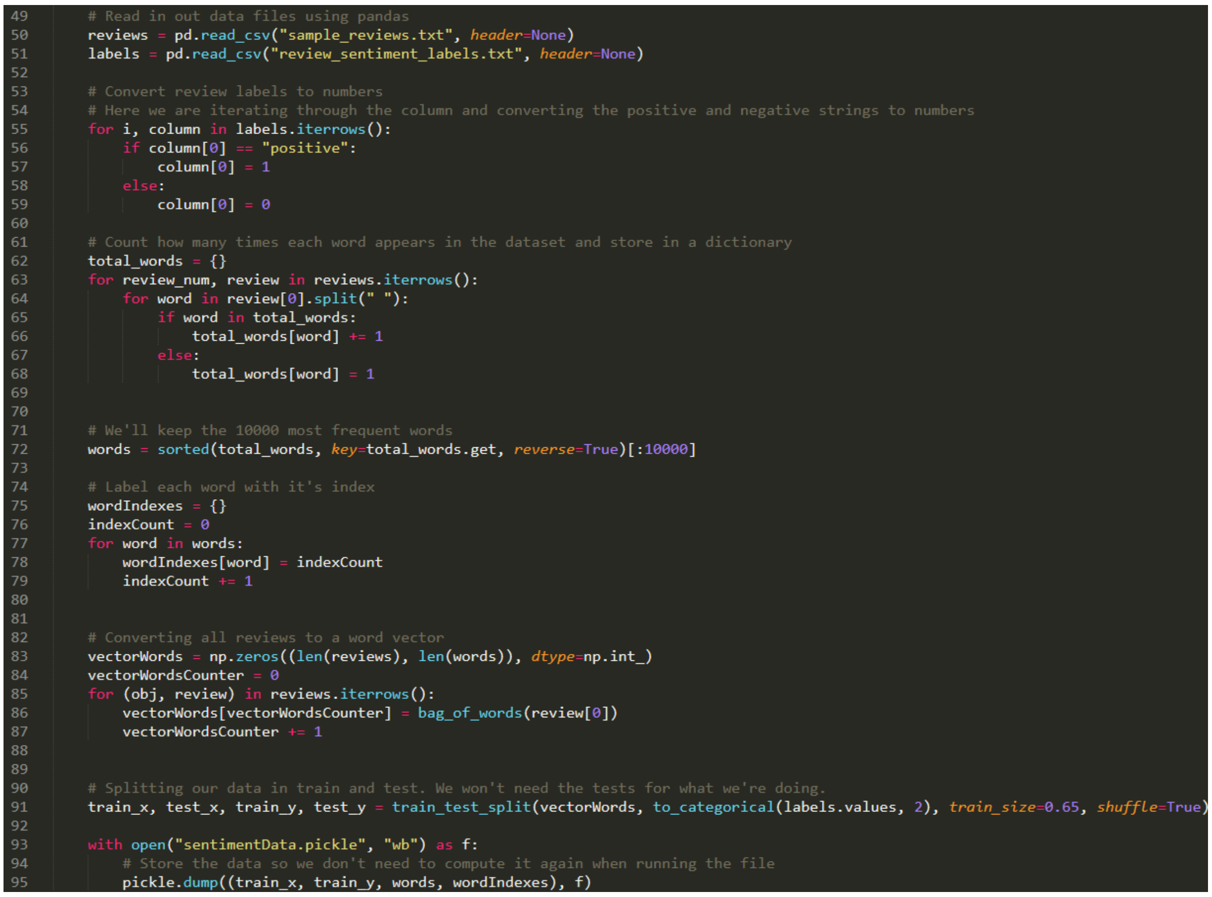
We followed a tutorial on how to format the datasets we used which can be found here:
https://notebook.community/kuo77122/deep-learning-nd/Lesson15-TFLearn/Sentiment%20Analysis%20with%20TFLearn%20-%20Solution and also in the appendix. We chose this tutorial because it uses TFlearn which we had previously used in the chatbot file whereas many other ways we had found online did it in completely different ways. Since we had used TFlearn before we wanted to continue using it with this file.

Once again, if a model is present in the directory, then these calculations will not be done again as that would be a waste of time and memory. Once we have a model, it is simply a case of that model making a prediction, much the same as the other model. With this model, we will receive two decimal numbers between 0 and 1. These decimal numbers represent the probability that the review belongs to a specific class (positive or negative). We choose the positive decimal and can work out the negative from that. Anything below 0.5 can be considered negative, whilst anything above 0.5 can be considered positive. So, if given a set of 10 reviews, and the bot returns 0.8, then we can say that the majority of what is said in those reviews is positive. For the user, the results will be displayed in the form of a bar chart and/or a pie chart, and we will convert the decimal number into a percentage.

Below you can see one such sample pie chart, courtesy of Python's built-in pie chart function. In this case, we tested one review and found it to be 99% positive.



Now we will look at some sample code from the sentiment analysis file where we are formatting our data so it may be used to train the sentiment model.



```
49      # Read in out data files using pandas
50      reviews = pd.read_csv("sample_reviews.txt", header=None)
51      labels = pd.read_csv("review_sentiment_labels.txt", header=None)
52  
53      # Convert review labels to numbers
54      # Here we are iterating through the column and converting the positive and negative strings to numbers
55      for i, column in labels.iterrows():
56          if column[0] == "positive":
57              column[0] = 1
58          else:
59              column[0] = 0
60  
61      # Count how many times each word appears in the dataset and store in a dictionary
62      total_words = {}
63      for review_num, review in reviews.iterrows():
64          for word in review[0].split(" "):
65              if word in total_words:
66                  total_words[word] += 1
67              else:
68                  total_words[word] = 1
69  
70  
71      # We'll keep the 10000 most frequent words
72      words = sorted(total_words, key=total_words.get, reverse=True)[:10000]
73  
74      # Label each word with it's index
75      wordIndexes = {}
76      indexCount = 0
77      for word in words:
78          wordIndexes[word] = indexCount
79          indexCount += 1
80  
81  
82      # Converting all reviews to a word vector
83      vectorWords = np.zeros((len(reviews), len(words)), dtype=np.int_)
84      vectorWordsCounter = 0
85      for (obj, review) in reviews.iterrows():
86          vectorWords[vectorWordsCounter] = bag_of_words(review[0])
87          vectorWordsCounter += 1
88  
89  
90      # Splitting our data in train and test. We won't need the tests for what we're doing.
91      train_x, test_x, train_y, test_y = train_test_split(vectorWords, to_categorical(labels.values, 2), train_size=0.65, shuffle=True)
92  
93      with open("sentimentData.pickle", "wb") as f:
94          # Store the data so we don't need to compute it again when running the file
95          pickle.dump((train_x, train_y, words, wordIndexes), f)
```

Line 50-51: Here we are simply reading in our test data, in this case a file of reviews and a file of the corresponding sentiments (positive and/or negative).

Line 55-59: Here we are converting the positive and negative sentiments in our labels file defined on line 61 into ones and zeros, the ones representing positive sentiment and zero representing negative sentiment. This is because we will be using ones and zeros in such a way later.

Line 62-72: Here we are simply counting the number of times each word appears in the dataset and storing the information in a dictionary. Then we will sort that dictionary by the 10000 most frequent words. We are cutting off those that only appear rarely because they will not have much impact on the overall system.

Line 75-79: Now we are making a new dictionary and assigning each word an index.

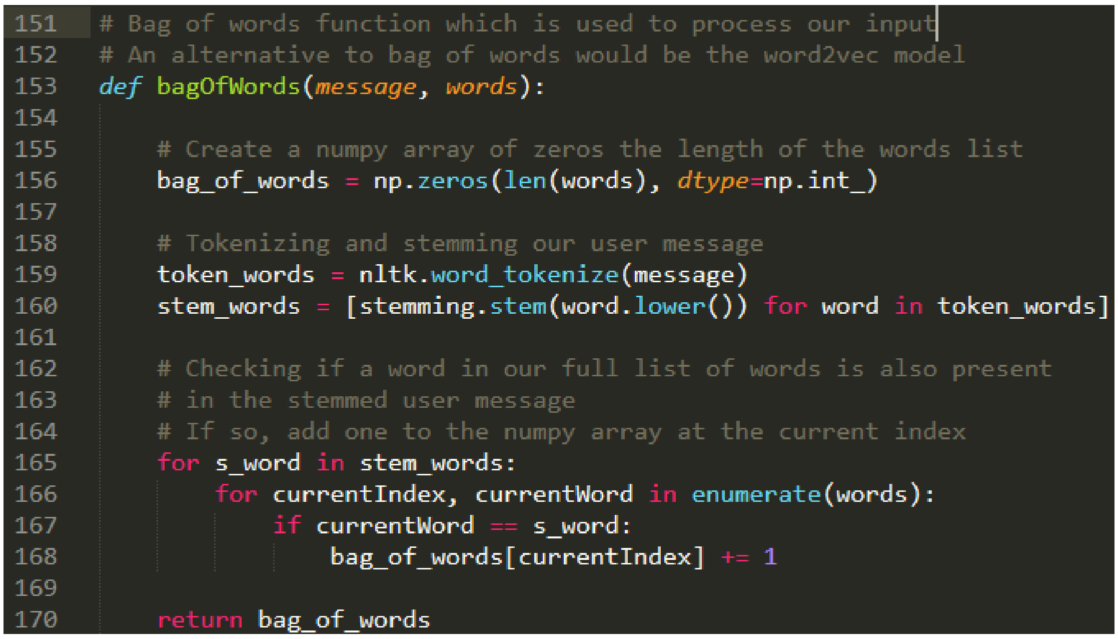
Line 83-87: Firstly, we define a numpy array with a specific shape. It is a list of lists, the length of our reviews dataset to cover all reviews, and the length of our words list to cover all words.

Then we iterate through our reviews, one review at a time (using a counter to move along index in our numpy array) using our bag of words function to convert each review into numbers. The numpy array is currently all zeros. If a word in the review is also present in the dataset, then we change the zero in the numpy array to a one. Once we get through all the reviews, our numpy array will be filled with lists containing ones and zeros.

Line 91: Now we need to split the data we have into training data. We use the *train_test_split* function from the sklearn library. This does the splitting of our array for us. We do not use the test portion of the split in this system, but the variables are needed to use the function. We are only interested in the training split. We are using the *to_categorical* function from tflearn which allows us to create a binary class matrix that we can use to classify the data into positive and negative. Our model will be trying to predict if a set of data (a review) belongs to one of two classes, positive or negative.

Line 93-95: We save the training data so we do not need to do the calculations each time the file is run. This is crucial as it takes quite some time for those calculations to be done.

Lastly, we will look at an implementation of the bag of words function. We use similar functions in both our chatbot file and sentiment file and it is necessary for allowing our models to make predictions.



```
151 # Bag of words function which is used to process our input
152 # An alternative to bag of words would be the word2vec model
153 def bagOfWords(message, words):
154 
155     # Create a numpy array of zeros the length of the words list
156     bag_of_words = np.zeros(len(words), dtype=np.int_)
157 
158     # Tokenizing and stemming our user message
159     token_words = nltk.word_tokenize(message)
160     stem_words = [stemming.stem(word.lower()) for word in token_words]
161 
162     # Checking if a word in our full list of words is also present
163     # in the stemmed user message
164     # If so, add one to the numpy array at the current index
165     for s_word in stem_words:
166         for currentIndex, currentWord in enumerate(words):
167             if currentWord == s_word:
168                 bag_of_words[currentIndex] += 1
169 
170     return bag_of_words
```

Line 156: Here we are defining a numpy array full of zeros and the length of our words list which contains all our words.

Line 159-160: Now we tokenize the user's message and stem the words into a list.

Line 165-170: Lastly, we loop through the stemmed words, then loop through our total words list and check if any word from the user message is also contained in the stemmed list, and if so, we add a one to our numpy array at that current index. Then we return the array at the end.

## 7. Problems Solved

There were numerous problems in our way during the making of this project. Thankfully, we were able to overcome most of them.

- Firstly, was the research. There was a lot of it in this project. There were not any tricks to get around this however, we just had to do it. After a while, we began noticing patterns and repetitions on some of the websites that we came across in how they did things such as training data, parsing data etc which helped us decide what to do ourselves. We have compiled a list of all the resources we have used in our Gitlab repository and in the appendix of this document.
- Another big issue we had was getting good accuracy on our models. This took some tinkering around and research to find out the best ways to increase the accuracy of any given model. For the chatbot model itself, we were well served with increasing the epoch number as this allowed the network to see the data lots more times and contributed to increasing the overall accuracy. However, for the sentiment analysis model, we could not just increase the epoch to help out with our accuracy problems. This is because the training data we are working with for the sentiment model is so much larger than that of the chatbot model that it simply takes too long to run. Therefore, we are forced to keep the epoch smaller.
- We did some user testing towards the end of the project. This meant we needed to get our website live on a server so users could use it without needing to run the code themselves and therefore install all of the necessary libraries.
- However, getting the website online took longer than expected. We had never deployed a flask server online before this. We tried some recommended hosting sites online but could get nothing working. We tried to host it on the DCU servers but quickly found that some of our libraries were simply too big and did not work there. We encountered the same problem when trying to host on Google Cloud Platform. Thanks to the help of a DCU staff member, we were pointed in the direction of Microsoft's Azure service. This still took

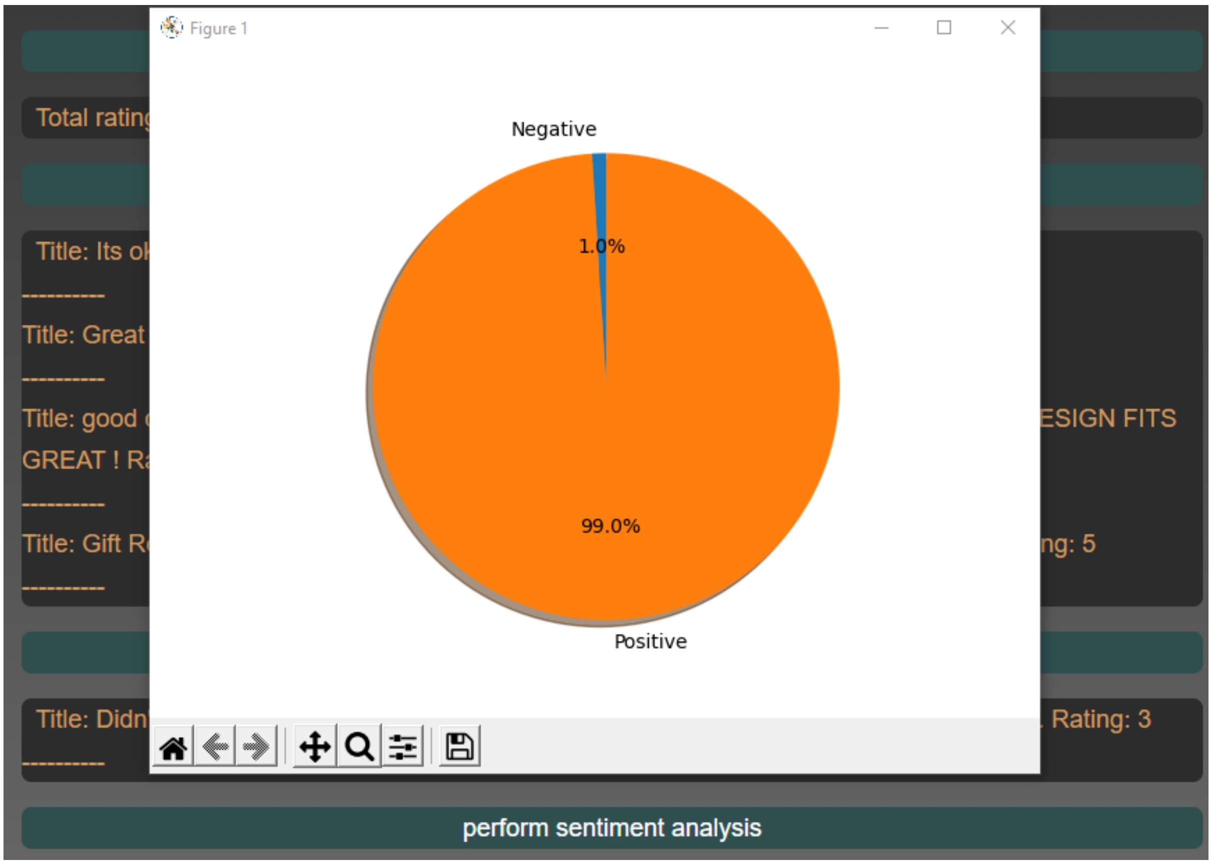
quite some time to figure out. Ultimately, we created a Linux Virtual machine with a server on it, and deployed the website there using Gunicorn (it does not work on Windows so we needed to use Linux). However, neither of us have a Linux machine so we needed to download a Ubuntu app and SSH into the virtual machine from there. After all that, at last, we got the website live so anyone with the URL can access and use it.

- There were also some problems we have had with our API's. We use an API to retrieve all the product information and reviews from Amazon. Initially, the API we used was very unreliable and would be down for maintenance for hours at a time which was simply not good for us. We have since switched to another API. This one works perfectly fine. There is an issue however. When getting reviews, the API can only retrieve the first page of reviews. A separate call is needed to the API for every subsequent page, which takes up time. We contacted the creators of the API to see if there was a way around this and, unfortunately, they said they could not do anything about it as this is only what Amazon allows.

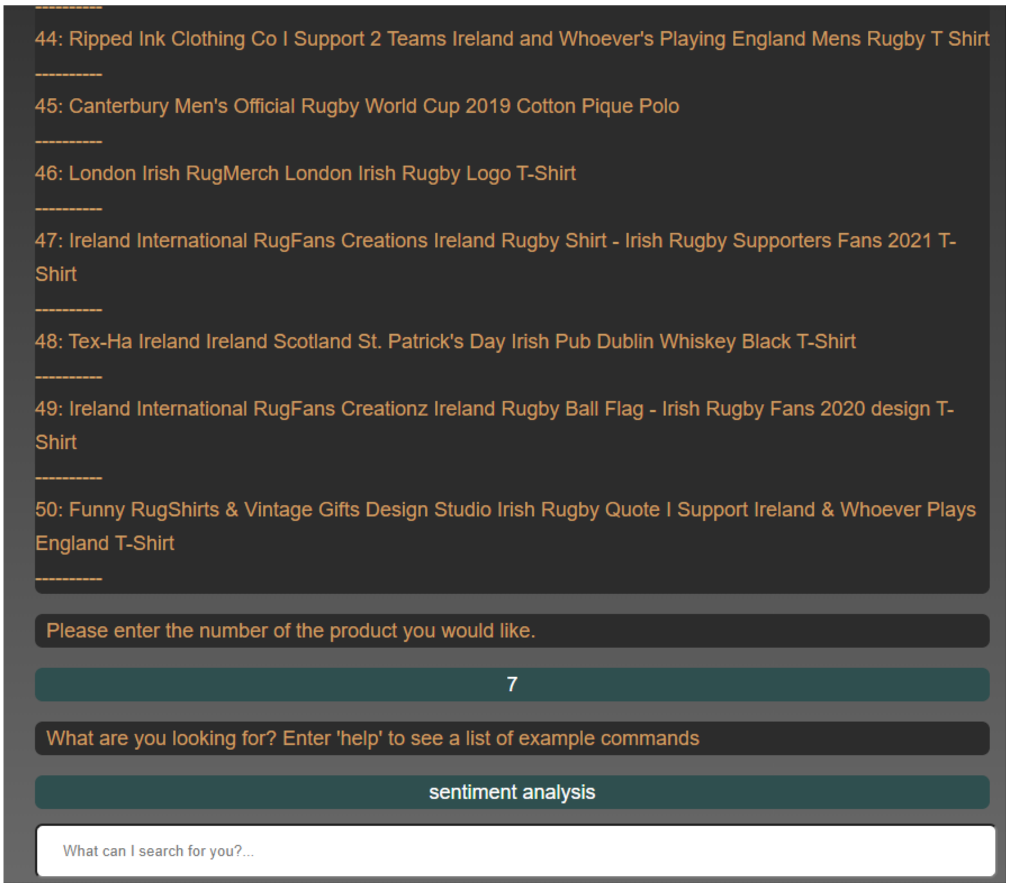
# 8. Results

All features we planned have been implemented and give the results that we expected however due to limitations with the API the speed of some commands are not as quick as we would like them to be. One case of this is when the user searches for a product. If there are a long number of search results the chatbot could take a few seconds to gather them and display them on screen to the user which is not ideal. However we feel like it is short enough that it will not badly affect their experience with the chatbot. From testing, the longest we have experienced is 2-3 seconds.

Another case is the sentiment analysis command, ideally we would have liked this to pull all reviews for a product all at once however, as stated earlier, the API creators were under certain limitations which also meant that we too were under those limitations in how many reviews we would get in at once. This meant that we had to implement it in a way that wasn't as efficient as we would have liked and had to limit products which we could perform it on as we had limited API calls and some products could use up a lot of these if we did sentiment analysis on them. This alternative approach works but takes a long time for products with a lot of reviews but due to this being a student product we cannot afford to pay for a larger subscription to the API that would make this approach usable.



This is a case where sentiment analysis works as the product only has 2 pages of reviews so sentiment analysis can be performed in quick time without using up many API calls.



```
----------
44: Ripped Ink Clothing Co I Support 2 Teams Ireland and Whoever's Playing England Mens Rugby T Shirt
----------
45: Canterbury Men's Official Rugby World Cup 2019 Cotton Pique Polo
----------
46: London Irish RugMerch London Irish Rugby Logo T-Shirt
----------
47: Ireland International RugFans Creations Ireland Rugby Shirt - Irish Rugby Supporters Fans 2021 T-Shirt
----------
48: Tex-Ha Ireland Ireland Scotland St. Patrick's Day Irish Pub Dublin Whiskey Black T-Shirt
----------
49: Ireland International RugFans Creationz Ireland Rugby Ball Flag - Irish Rugby Fans 2020 design T-Shirt
----------
50: Funny RugShirts & Vintage Gifts Design Studio Irish Rugby Quote I Support Ireland & Whoever Plays England T-Shirt
----------
Please enter the number of the product you would like.
7
What are you looking for? Enter 'help' to see a list of example commands
sentiment analysis
What can I search for you?...
```

This is a case where the sentiment analysis doesn't work well because of API limitations. The chatbot takes a very long time to collect all the reviews and is not an ideal option and will result in users waiting a long time before getting results.

All other commands work quickly and as we intended. We feel that all these features are implemented the best way possible with the time and resources we had and gives the user the best experience when using our chatbot to search for products.

# 9. Future work

For future work, there are a few options. At the moment, our chatbot only works on Amazon.co.uk. We could possibly expand this to give the user the choice of a number of different online retailers such as Ebay, Walmart, etc. This would greatly increase the user base of the system as not everyone shops on Amazon despite it being the largest online retailer in the world.

This would be possible to do as our method of sentiment analysis would not change. We would just need to acquire product information and reviews from the other sites which could likely be done in the same manner as we did with Amazon, through an API.

If we wanted to as well, we could try different methods of training models. Tflearn is not the only way to go about it. We came across numerous other ways during our research. This could help us figure out which methods are best, or which are better or worse in one area or another.

A less important addition would be some additional dialogue options with the bot itself. At the moment, excluding all it does with the products on Amazon, it responds to basic greetings. We could add to this to allow it to have deeper conversations with the user. This is a bells and whistles change. It adds more personality to the bot, but does not affect the main experience the user's will be having which is centred around Amazon products and what they can do with them.

## 10. Appendix

Below is a list of sources that we used for research and/or acquired code from:

https://usefulangle.com/post/85/css-typewriter-animation

https://matplotlib.org/stable/gallery/pie_and_polar_charts/pie_features.html

https://www.tutorialspoint.com/matplotlib/matplotlib_bar_plot.htm

https://datatofish.com/replace-values-pandas-dataframe/

https://www.techwithtim.net/tutorials/ai-chatbot/part-1/

https://codinginfinite.com/chatbot-in-python-flask-tutorial/

https://www.freecodecamp.org/news/an-introduction-to-bag-of-words-and-how-to-code-it-in-python-for-nlp-282e87a9da04/

https://numpy.org/doc/stable/reference/generated/numpy.reshape.html

http://tflearn.org/data_utils/\#to_categorical

https://chatbotsmagazine.com/contextual-chat-bots-with-tensorflow-4391749d0077

http://tflearn.org/

https://realpython.com/sentiment-analysis-python/

https://realpython.com/python-nltk-sentiment-analysis/

https://keras.rstudio.com/reference/fit.html

https://notebook.community/kuo77122/deep-learning-nd/Lesson15-TFLearn/Sentiment%20Analysis%20with%20TFLearn%20-%20Solution

https://levelup.gitconnected.com/natural-language-processing-and-sentiment-analysis-using-tensorflow-c2948f2623f

https://app.rainforestapi.com/login

https://www.datacamp.com/community/tutorials/simplifying-sentiment-analysis-python

https://blog.devgenius.io/training-an-ml-model-for-sentiment-analysis-in-python-63b6b8c68792

http://tflearn.org/models/dnn/\#deep-neural-network-model

https://www.programmersought.com/article/90993717162/

https://medium.com/dev-genius/training-an-ml-model-for-sentiment-analysis-in-python-63b6b8c68792

https://towardsdatascience.com/how-to-build-a-basic-chatbot-from-scratch-f63a2ccf5262

https://www.javatpoint.com/how-to-create-text-box-in-html

https://www.programiz.com/python-programming/json

https://stackabuse.com/reading-and-writing-json-to-a-file-in-python/

http://tflearn.org/tutorials/quickstart.html\#training